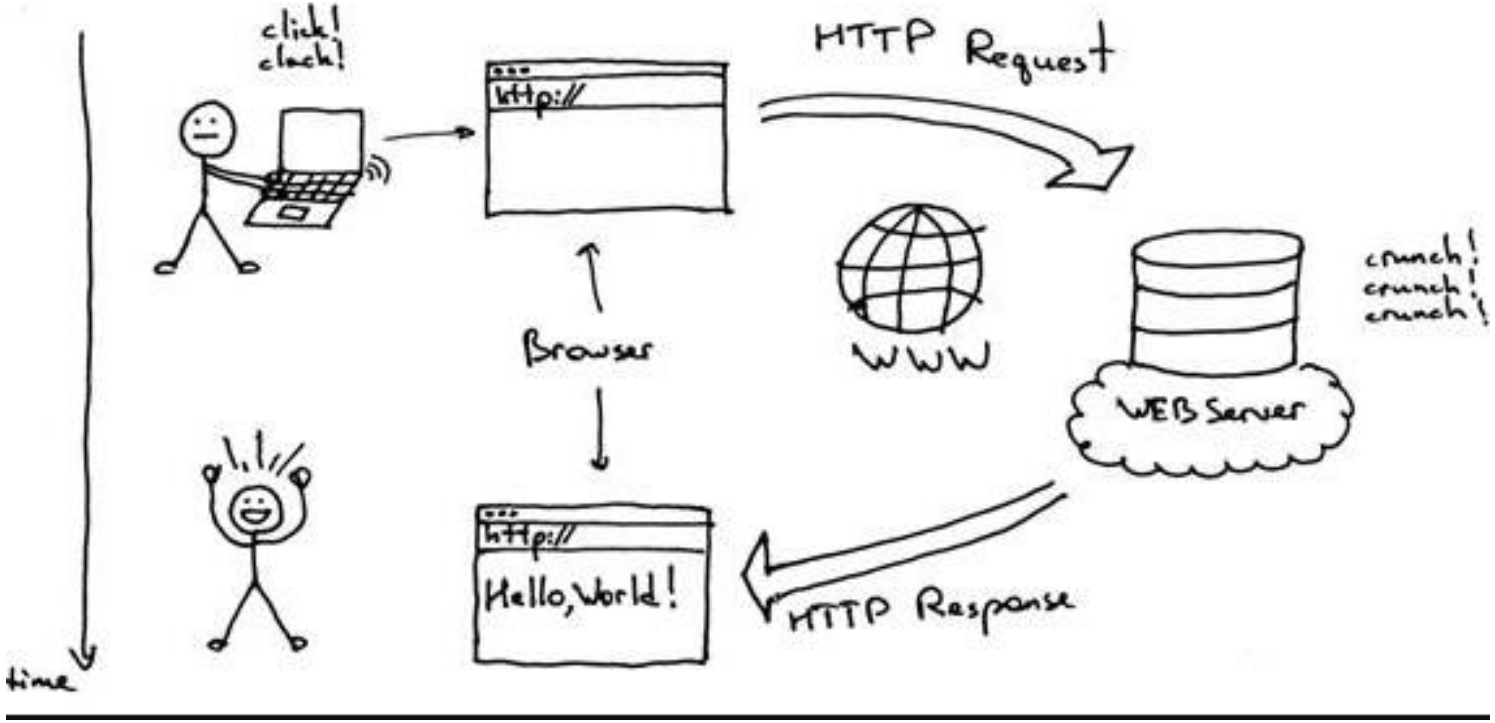
Treinamento Node.js

Web Server

2f48696768207065726f726d616e63652e2044656c6976657265642e2f48696768207065726f726d616e63652e2044656c6976657265642e2f48696768207065726f726d616e63652e2044656c6976657265642e2f48696768207065726f726d616e63652e2044656c6976657265642e2f48696768207065726f726d616e63652e2044656c6976657265642e2f48696768207065726f726d616e63652e2044656c6976657265642e2f48696768207



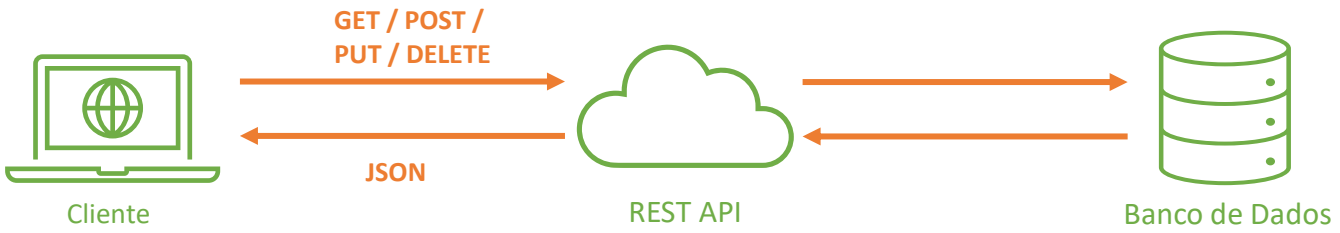
API RESTful

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

- Uma API (“Application Programming Interface”) - Interface de Programação de Aplicações é um conjunto de definições e protocolos, usado no desenvolvimento e na integração de aplicações.
- Funciona como um mediador, ou comunicador, entre o usuário e o sistema

- REST é a sigla em inglês para "Representational State Transfer", que em português significa transferência de estado representacional.

- API RESTful, é uma interface de programação de aplicações em conformidade com as restrições do estilo de arquitetura REST, permitindo a interação com serviços web RESTful.



API RESTful

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

- Cada resposta que a aplicação REST retorna, é enviado um código definindo o status da requisição. Por exemplo:

  - 200 (OK), requisição atendida com sucesso;
  - 201 (CREATED), objeto ou recurso criado com sucesso;
  - 204 (NO CONTENT), objeto ou recurso deletado com sucesso;
  - 400 (BAD REQUEST), ocorreu algum erro na requisição (podem existir inumeras causas);
  - 404 (NOT FOUND), rota ou coleção não encontrada;
  - 500 (INTERNAL SERVER ERROR), ocorreu algum erro no servidor.

API RESTful

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

- As requisições HTTP são compostas de verbos (ou métodos) que indicam o que deve ser feito. Dentre esses verbos, há quatro que são os mais utilizados:

  - GET: Buscar informações do backend
  - POST: Criar uma informação no backend;
  - PUT: Alterar uma informação no backend.
  - DELETE: Deletar uma informação no backend;

API RESTful

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

Tipos de Parâmetros

- **Query Params** (também chamados de parâmetros GET): são utilizados principalmente para filtros e paginação

  http://localhost:3333/users?name=João&age=30

- **Route Params**: usado para identificar recursos nas rotas PUT e DELETE (Atualizar/Deletar).

  http://localhost:3333/users/2

- **Request Body**: utilizado quando o cliente envia algum conteúdo (um formulário de criação de usuário, por exemplo). Ele pode ser usado tanto na rota de criação (POST) quanto na edição de um recurso (PUT). Essas informações chegam no backend por meio de JSON.

Express

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

Método de Resposta

- Os métodos de resposta (res) podem enviar uma resposta ao cliente, e finalizar o ciclo solicitação-resposta.
  https://expressjs.com/pt-br/4x/api.html

Método	Descrição
res.download()	Solicita que seja efetuado o download de um arquivo
res.end()	Termina o processo de resposta.
res.json()	Envia uma resposta JSON.
res.jsonp()	Envia uma resposta JSON com suporta ao JSONP.
res.redirect()	Redireciona uma solicitação.
res.render()	Renderiza um modelo de visualização.
res.send()	Envia uma resposta de vários tipos.
res.sendFile	Envia um arquivo como um fluxo de octeto.
res.sendStatus()	Configura o código do status de resposta e envia a sua representação em sequência de caracteres como o corpo de resposta.

MySQL

2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207

```
1   var mysql = require('mysql');
2
3   var con = mysql.createConnection({
4     host: "localhost",
5     user: "root",
6     password: "root",
7     database: "mydb"
8   });
9
10  con.connect(function(err) {
11    if (err) throw err;
12    console.log("Connected!");
13  });
```

MySQL

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

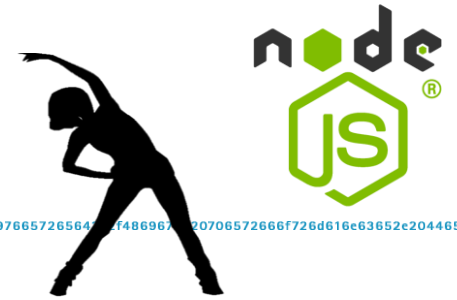
```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "root",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  //Select all customers and return the result object:
  con.query("SELECT * FROM aluno", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

MySQL

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

```
10  con.connect(function(err) {
11    if (err) throw err;
12    console.log("Connected!");
13    //Make SQL statement:
14    var sql = "INSERT INTO aluno
15       (idAluno, nome, tipo_aluno, tipoBolsa, valorMensalidade) VALUES ?";
16    //Make an array of values:
17    var values = [
18      ['101','John',' ','B50',500],
19      ['102','Peter',' ','B90',100],
20      ['103','Chuck',' ',' ',1000],
21      ['104','Viola',' ','B10',900]
22    ];
23    //Execute the SQL statement, with the value array:
24    con.query(sql, [values], function (err, result) {
25      if (err) throw err;
26      console.log("Number of records inserted: " + result.affectedRows);
27    });
28  });
```

Atividade

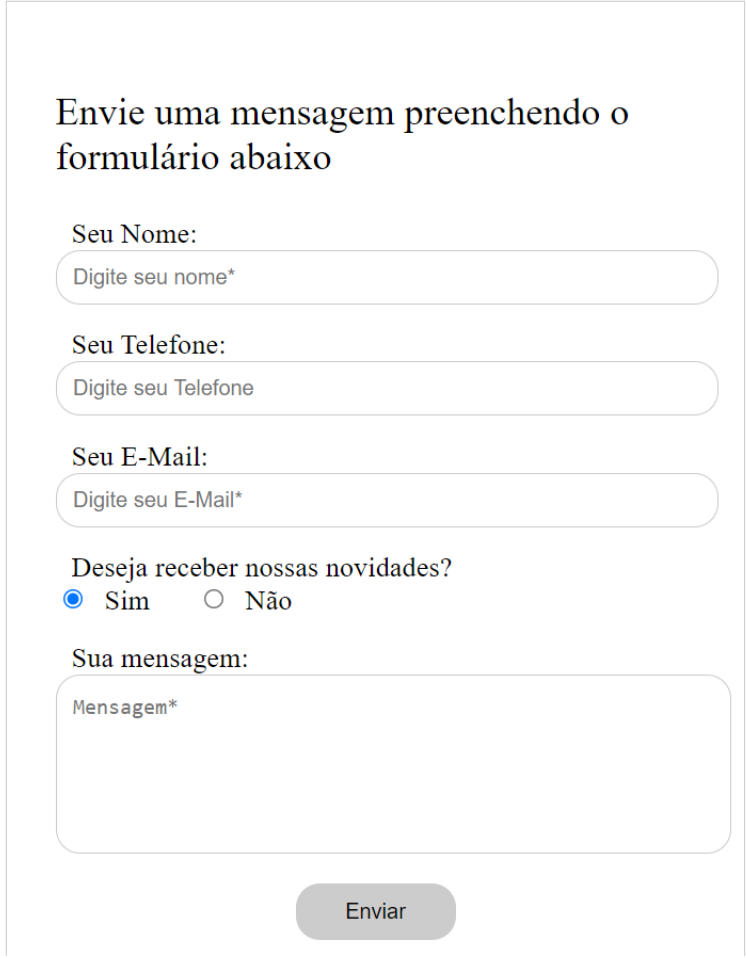


- Criar uma tabela usuários no MySql.

```
CREATE TABLE `user` (
  `id` smallint unsigned NOT NULL AUTO_INCREMENT,
  `nome` varchar(200) NOT NULL,
  `telefone` varchar(15) DEFAULT NULL,
  `email` varchar(150) DEFAULT NULL,
  `novidades` tinyint(1) NOT NULL,
  `mensagem` text,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

- Criar a API que vai receber os dados enviados do formulário.
- Salvar os dados na tabela.

  - PassagemDeValores.html
  - Node_Sparametro2.js



AXIOS

2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207065726f726d616e63652c2044656c6976657265642c2f48696768207

O que é o Axios?

- Axios é um cliente HTTP baseado em promises para o browser e Node. js.
- O Axios facilita o envio de solicitações HTTP assíncronas para endpoints REST e a execução de operações CRUD.

```
axios.get(...)
    .then(...)
    .catch(error => {
        // what now?
    })
```

AXIOS

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

```
axios.post('/user_login', {
    username: ‘peterParker',
    password: 'spiderman’
})
.then(function (response) {
    console.log(response);
})
.catch(function (error) {
    console.log(error);
});
```

AXIOS

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
axios.get("/list")
    .then(function (response) {
        const lista = response[0].data
        console.log(lista);
    })
    .catch(function (error) {
        console.log(error);
    });
```

Atividade

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070

Crie uma API Axios para consumir a API VIACEP.

- Criar uma página web simples para informar o CEP.
- Criar a tabela de CEP no MySql.
- Fazer GET na viacep.
- Recuperar os dados.
- Conectar no mySQL.
- Salvar em uma tabela o cep recuperado.
- https://viacep.com.br/ws/01001000/json/

```
CREATE TABLE IF NOT EXISTS `cep` (
    `cep` VARCHAR(9) PRIMARY KEY,
    `logradouro` VARCHAR(300) NOT NULL,
    `complemento` VARCHAR(200) NULL,
    `bairro` VARCHAR(200) NOT NULL,
    `localidade` VARCHAR(200) NOT NULL,
    `uf` VARCHAR(2) NOT NULL,
    `ibge` VARCHAR(10) NOT NULL,
    `gia` VARCHAR(200) NULL,
    `ddd` VARCHAR(3) NOT NULL,
    `siafi` VARCHAR(10) NOT NULL
) ENGINE=InnoDB;
```

Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

- JavaScript fornece uma maneira fácil de escapar de um inferno de callback. Isso é feito por **promises**.

- Uma **promise** é um objeto retornado de qualquer função assíncrona, ao qual métodos de retorno de chamada podem ser adicionados com base no resultado da função anterior.

- As **promises** usam o método **.then()** para tratar retornos de chamada assíncronos.

- Podemos encadear quantas chamadas de retorno quisermos e a ordem também é estritamente mantida.

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
getData(a ⇒ {
    getMoreData(a, b ⇒ {
        getMoreData(b, c ⇒ {
            getMoreData(c, d ⇒ {
                getMoreData(d, e ⇒ {
                    console.log(e);
                });
            });
        });
    });
});
```

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

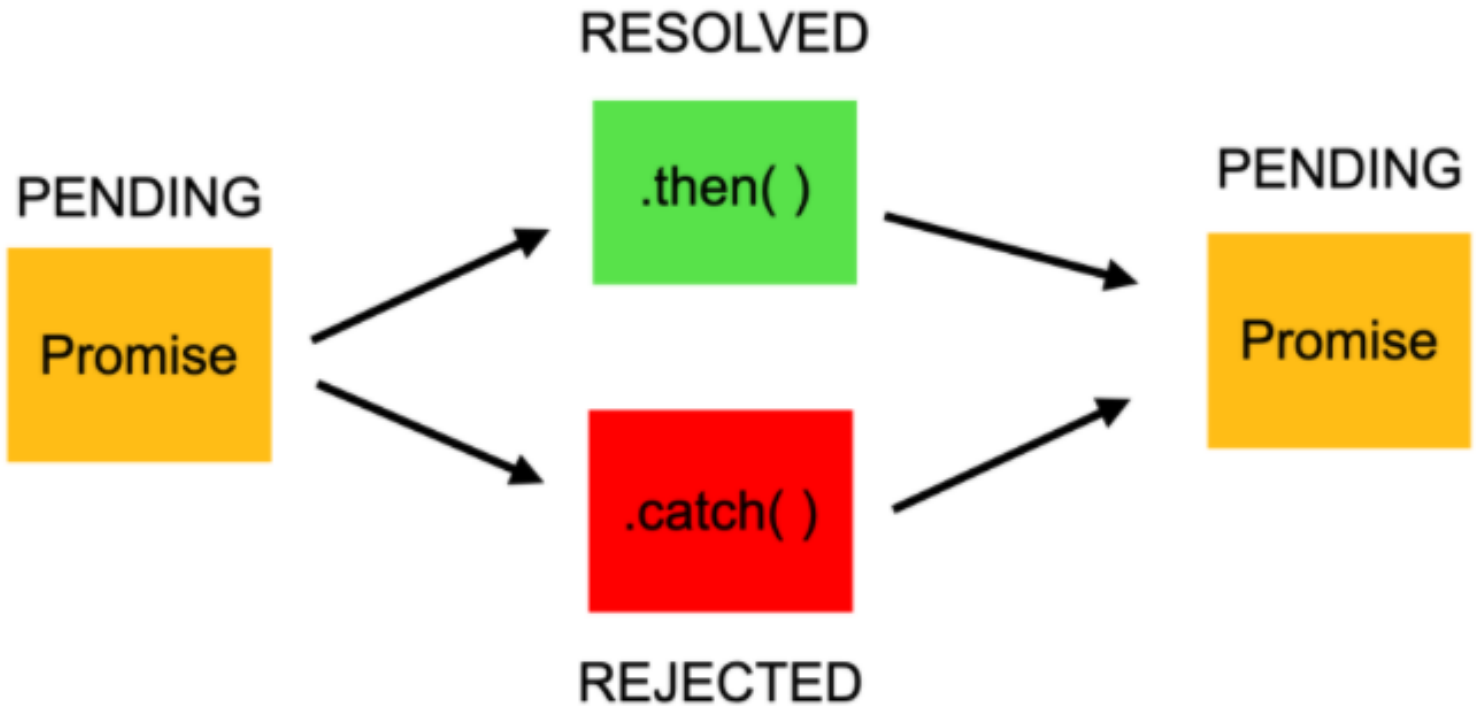
- As **promises** usam o método **.fetch()** para buscar um objeto da rede.
  Ele também usa o método **.catch()** para capturar qualquer exceção quando qualquer bloco falhar.

- Portanto, essas **promises** são colocadas na fila de eventos para que não bloqueiem o código JS subsequente. Além disso, quando os resultados são retornados, a fila de eventos termina suas operações.

- Existem também outras palavras-chave e métodos úteis, como **async**, **wait**, **settimeout()** para simplificar e fazer melhor uso dos retornos de chamada.

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207



Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070

- **Promises** vem do termo “promessa”, que representa um valor que pode estar disponível em algum momento no futuro, no presente, ou nunca estar disponível.

- Ele é um objeto utilizado em processamento **assíncrono**.

- Um **promise** representa um proxy para um valor não necessariamente conhecido, permitindo uma associação de métodos de tratamento para eventos em uma ação assíncrona na hipótese de sucesso ou falha, permitindo que o método assíncrono retorne uma “promessa” ao valor em algum momento no futuro.

- Elas não eram nativas do **JavaScript** até o **ES6**, quando houve uma implementação oficial na linguagem, antes delas, a maioria das funções usavam ***callbacks***.

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

Estados de uma Promise

- *Pending:* O estado inicial da Promise, ela foi iniciada mas ainda não foi realizada nem rejeitada
- *Fulfilled:* Sucesso da operação, é o que chamamos de uma Promise **realizada** (ou, em inglês, *resolved)* — eu, pessoalmente, prefiro o termo **resolvida**.
- *Rejected:* Falha da operação, é o que chamamos de uma Promise **rejeitada** (em inglês, *rejected*)

Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

- NODE_PromisessoPar.js
- NODE_PromisessoPares.js
- NODE_PromisessoParesTeste.js

```
soAceitaPares(2)
    .then(result => console.log("Promise resolved: " + result))
    .catch(error => console.log("Promises rejected: " + error));

console.log("teste");
```

```
//index.js
function soAceitaPares(numero){
    const promise = new Promise( (resolve, reject) => {
                        if (numero % 2 === 0) {
                            const resultado = 'Viva, é par!';
                            resolve(resultado);
                        }
                        else {
                            reject(new Error("Você passou um número ímpar!"));
                        }
                    });
    return promise;
}
```

Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

- NODE_Promisedivide.js
- NODE_PromisedivideTeste.js

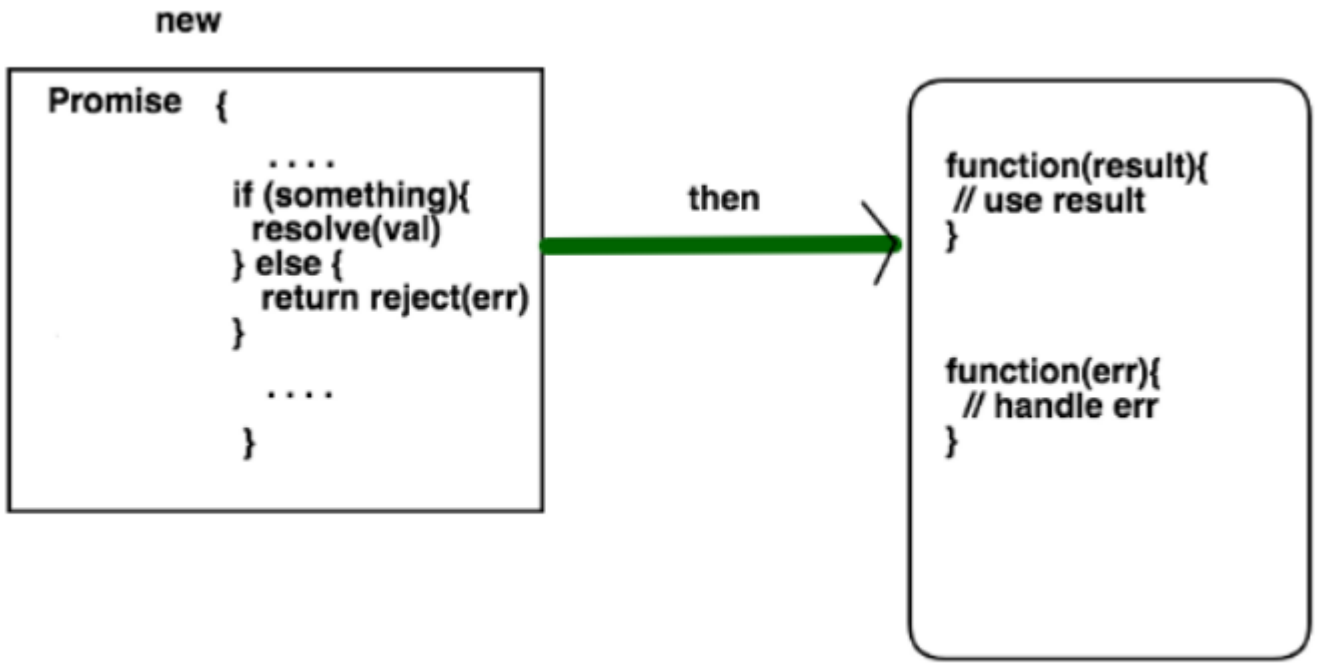
```
1  function dividePelaMetade(numero){
2      if(numero % 2 !== 0)
3          return Promise.reject(new Error("Não posso dividir um número ímpar!"));
4      else
5          return Promise.resolve(numero / 2);
6  }
```

```
1  if(soAceitaPares(numero))
2     dividePelaMetade(numero);
```

```
1  const numero = 2;
2  soAceitaPares(2)
3      .then(result => dividePelaMetade(numero))
4      .then(result2 => console.log("A metade de " + numero + " é " + result2))
5      .catch(error => console.log("Promises rejected: " + error));
6
7  console.log("teste");
```

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207



What happens if you try to access the value from promise before it is

Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

```
// Criando uma promise
const p = new Promise((resolve, reject) => {
  try {
    resolve(funcaoX())
  } catch (e) {
    reject(e)
  }
})

// Executando uma promise
p
  .then((parametros) => /* sucesso */)
  .catch((erro) => /* erro */)

// Tratando erros e sucessos no then
p
  .then(resposta => { /* tratar resposta */ }, erro => { /* tratar erro */ })
```

Promise

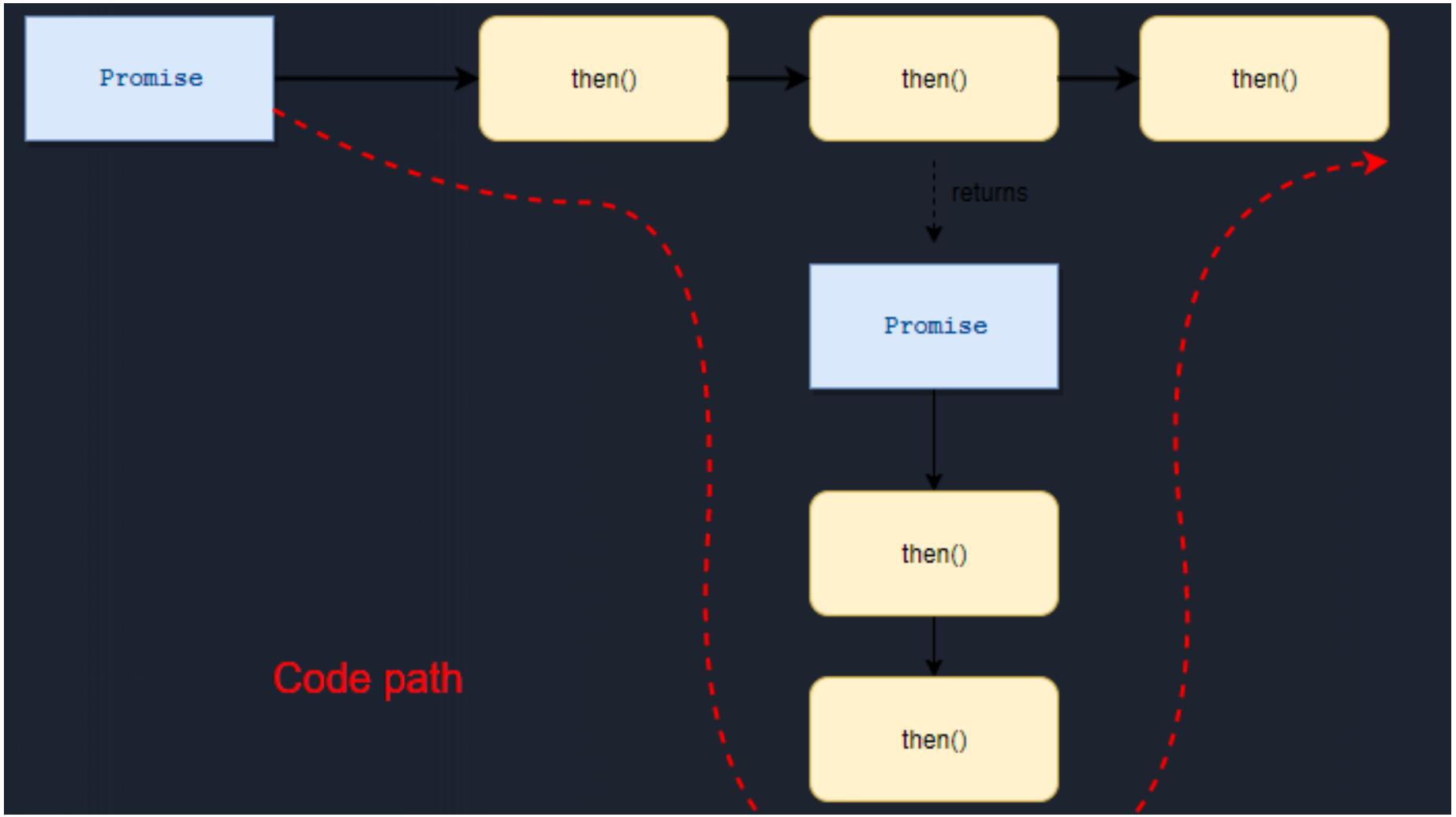
2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

```
const p = new Promise((resolve, reject) => {
  if (Math.random() > 0.5) resolve('yay')
  reject('no')
})

p
.then(function acao1 (res) { console.log(`${res} da ação 1`); return res; })
.then(function acao2 (res) { console.log(`${res} da ação 2`); return res; })
.then(function acao3 (res) { console.log(`${res} da ação 3`); return res; })
.catch(function erro (rej) { console.error(rej) })
```

NODE_promise.js

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207



Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

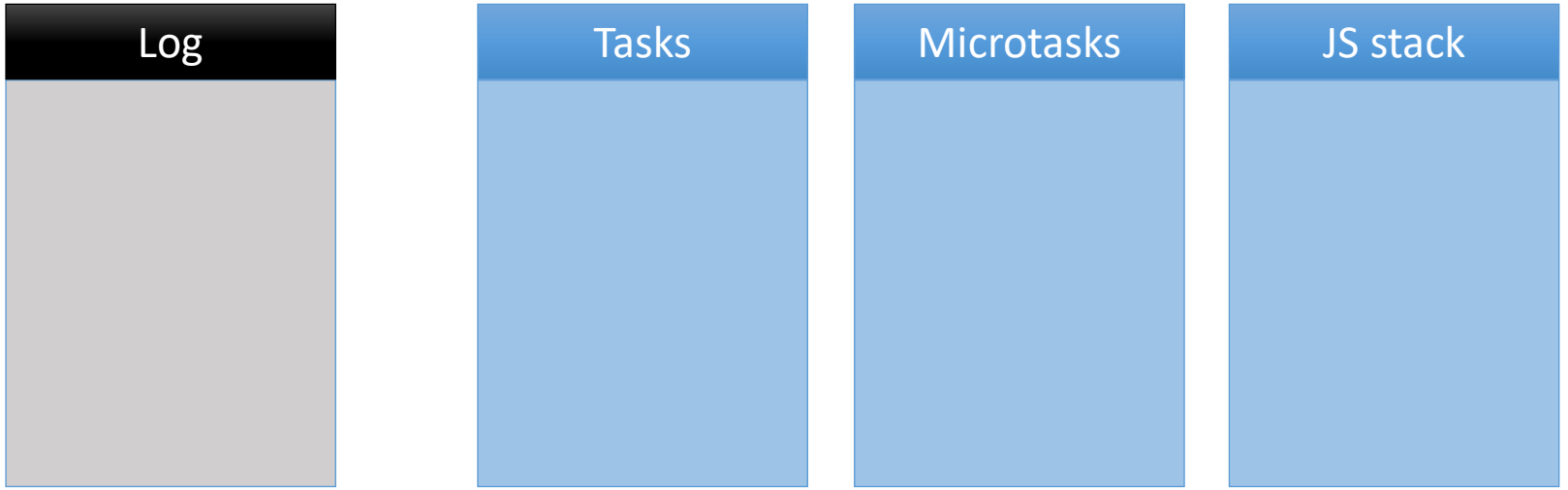
```
1  const innerPromise = (val) => Promise.resolve(console.log(val + 1))
2                                .then(() => console.log(val + 2))
3                                .then(() => {
4                                    console.log(5)
5                                    return 5
6                                })
7
8  const chain = Promise.resolve(console.log(0))
9  .then(() => console.log(1))
10 .then(() => console.log(2) || 2)
11 .then(innerPromise)
12 .then((five) => console.log(five + 1))
13
```

NODE_promisses3.js

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

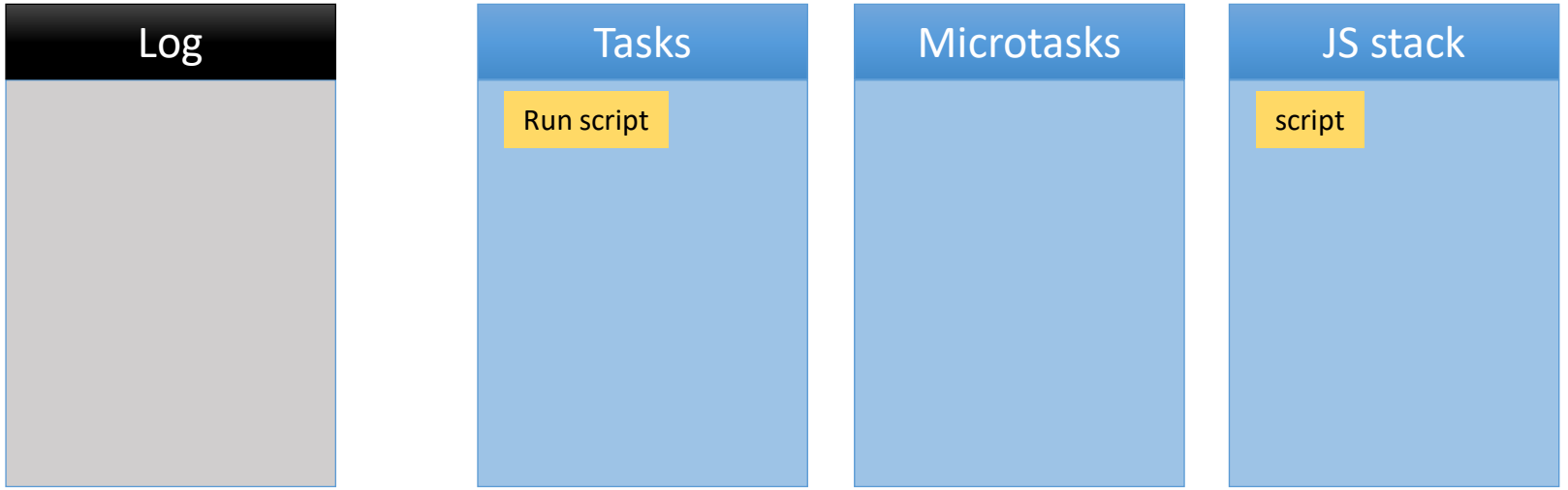
```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```



Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

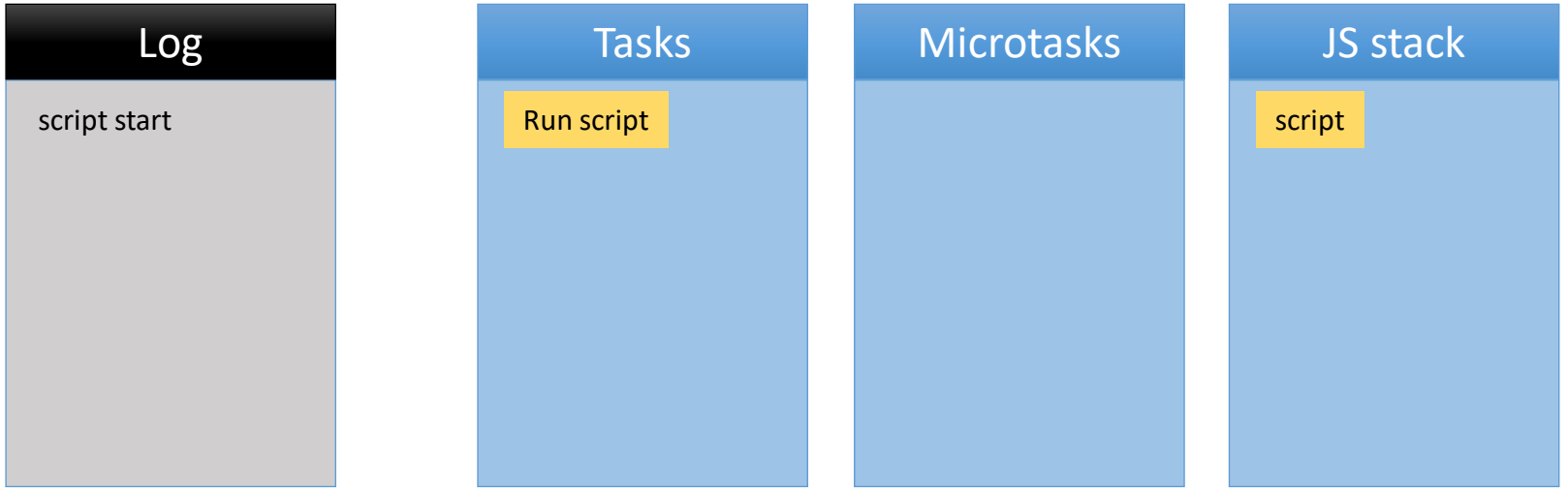
```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```



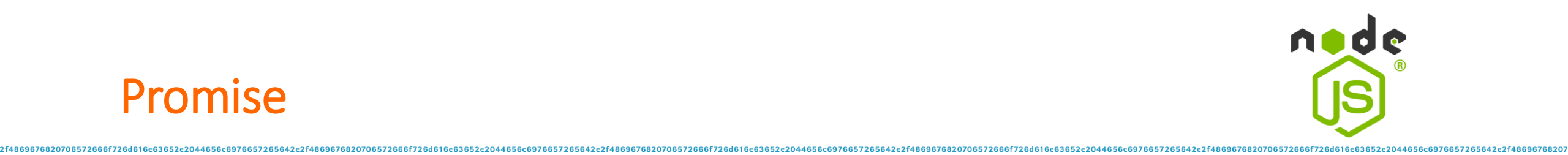
Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

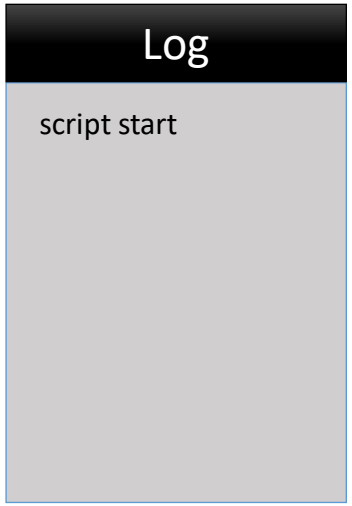
```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

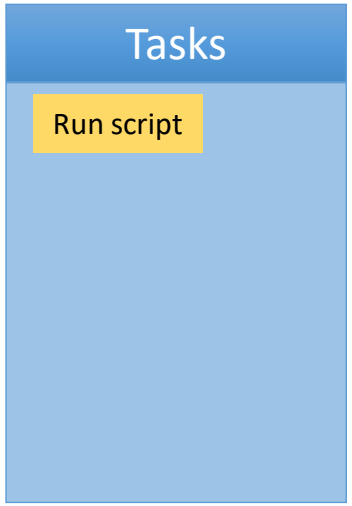


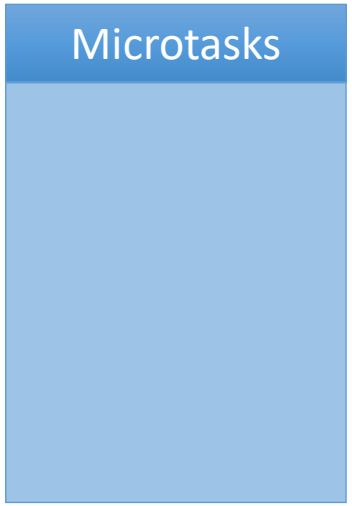
Promise

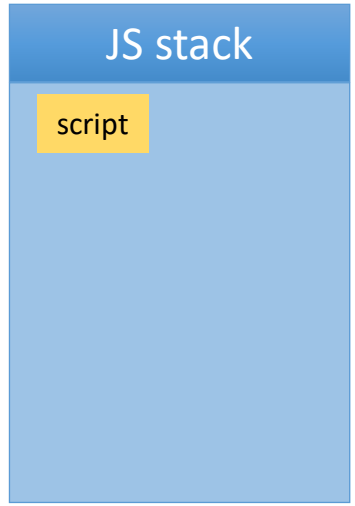


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```



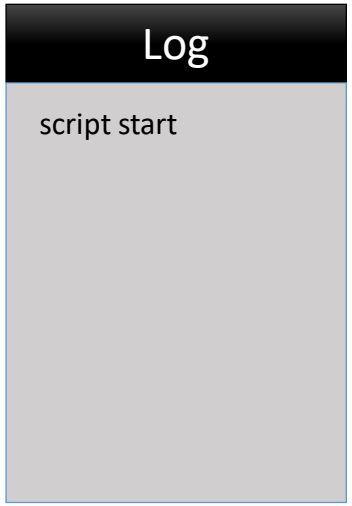


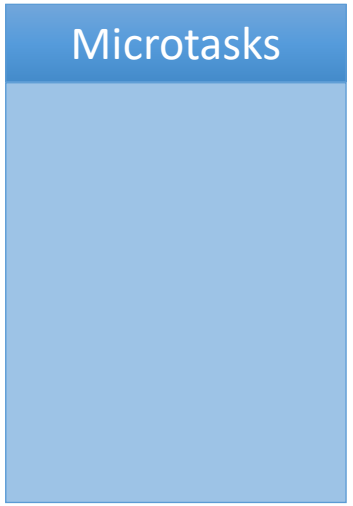


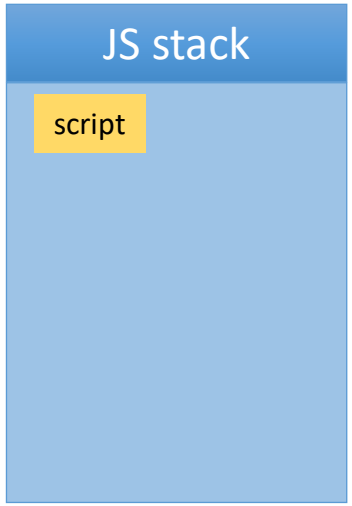


Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

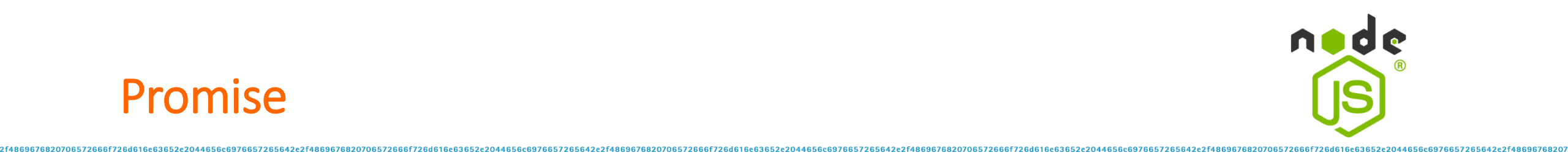




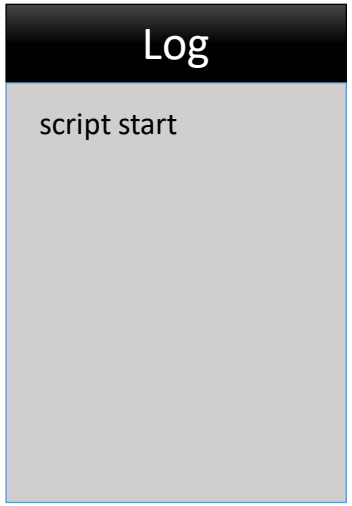


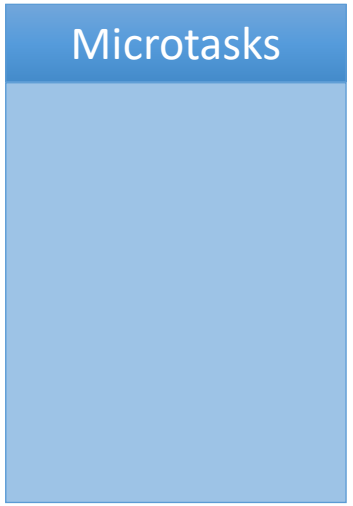
Callbacks do setTimeout são enfileirados como tarefas

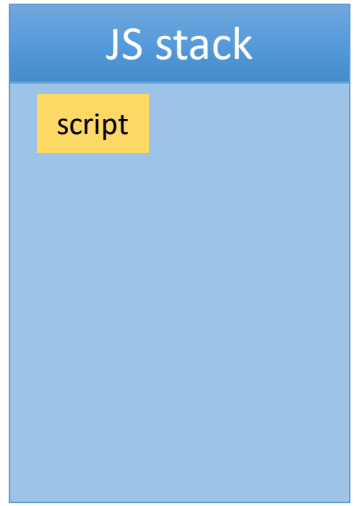
Promise



```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```



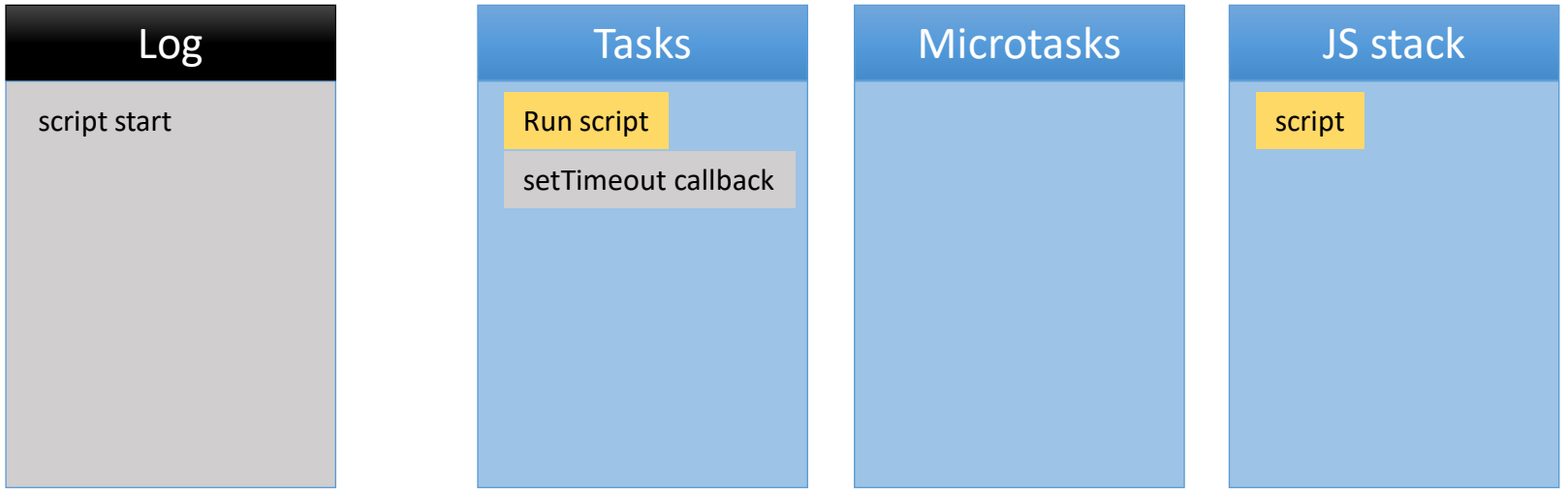




Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

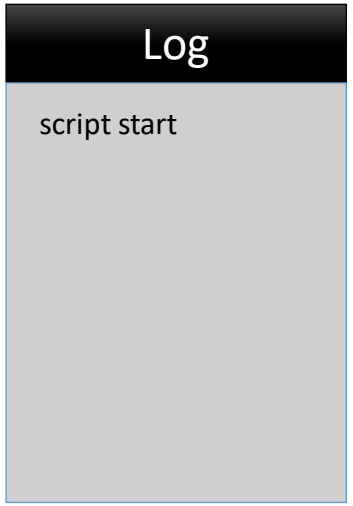
```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

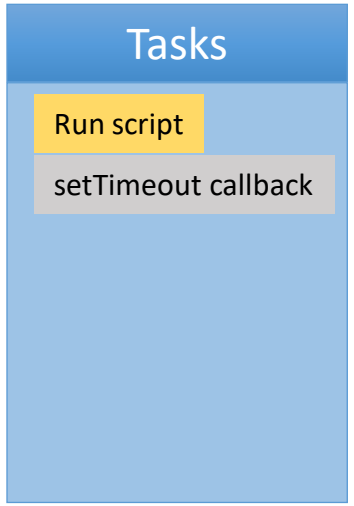


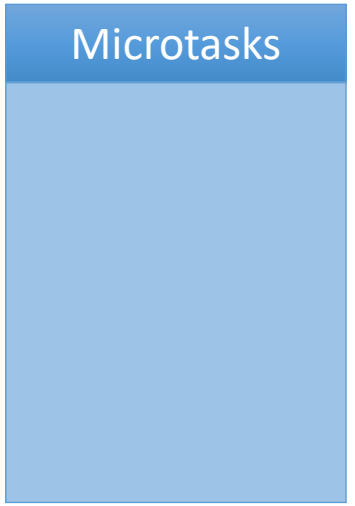
Callbacks de Promises são enfileirados como microtasks

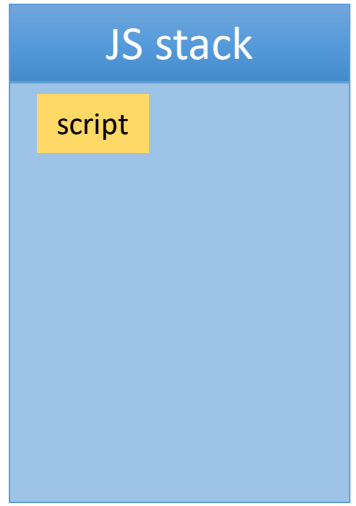
Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```









Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack
script

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack
script

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log	Tasks	Microtasks	JS stack
script start	Run script	Promise then	
script end	setTimeout callback		

No final de uma tarefa, nós processamos as microtasks

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack
Promise callback

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks	Microtasks	JS stack
Run script	Promise then	Promise callback
setTimeout callback		

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack
Promise callback

Este callback de Promise retorna 'undefined', o que enfileira o próximo callback de Promise como uma microtask

Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks
Run script
setTimeout callback

Microtasks
Promise then
Promise then

JS stack
Promise callback

Promise

2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f486967682070657266f726d616e63652c2044656c6976657265642c2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks
Run script
setTimeout callback

Microtasks
Promise then
Promise then

JS stack

Esta microtask está concluída, então seguimos para a próxima na fila

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks
Run script
setTimeout callback

Microtasks
Promise then
Promise then

JS stack

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1

Tasks
Run script
setTimeout callback

Microtasks
Promise then

JS stack
Promise callback

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
Run script	Promise then	Promise callback
setTimeout callback		

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
Run script		
setTimeout callback		

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log	Tasks	Microtasks	JS stack
script start	Run script		
script end	setTimeout callback		
promise1			
promise2			

E essa tarefa está concluída! O navegador pode atualizar a renderização.

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
Run script		
setTimeout callback		

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
Run script		
setTimeout callback		

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
setTimeout callback		

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
setTimeout callback		

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2

Tasks	Microtasks	JS stack
setTimeout callback		setTimeout callback

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2
setTimeout

Tasks	Microtasks	JS stack
setTimeout callback		setTimeout callback

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2
setTimeout

Tasks	Microtasks	JS stack
setTimeout callback		

Promise


```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2
setTimeout

Tasks

Microtasks

JS stack

Promise

2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f486967682070657266f726d616e63652e2044656c6976657265642e2f48696768207

```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve()
  .then(function () {
    console.log('promise1');
  })
  .then(function () {
    console.log('promise2');
  });

console.log('script end');
```

Log
script start
script end
promise1
promise2
setTimeout

Tasks

Microtasks

JS stack

Fim